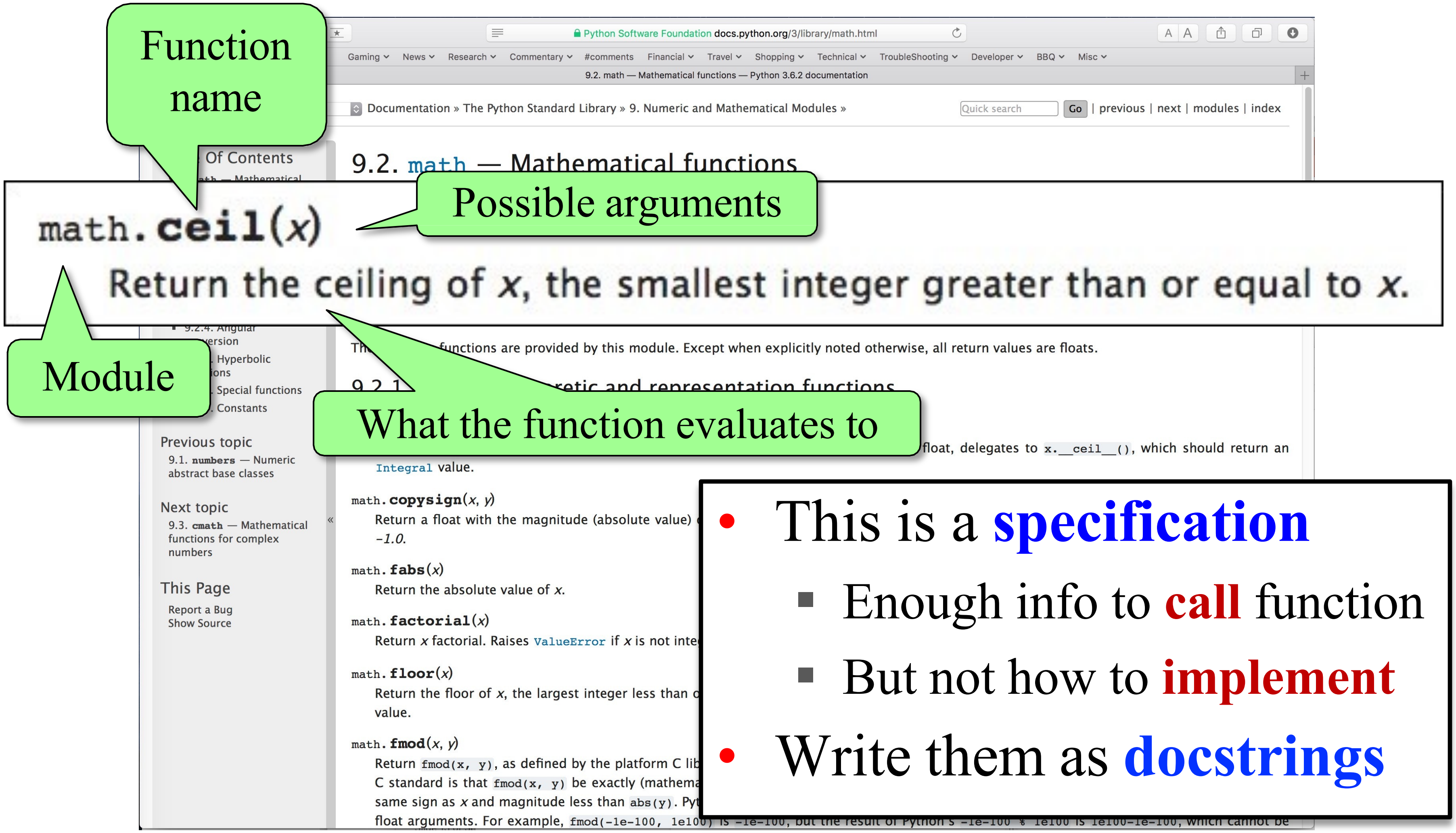
Lecture 04

Specifications & Testing

Recall: The Python API



Function name
Possible arguments
Module
What the function evaluates to
Python Software Foundation docs.python.org/3/library/math.html
Gaming News Research Commentary #comments Financial Travel Shopping Technical TroubleShooting Developer BBQ Misc
9.2. math — Mathematical functions — Python 3.6.2 documentation

Documentation » The Python Standard Library » 9. Numeric and Mathematical Modules »

Quick search Go | previous | next | modules | index

Of Contents
ath — Mathematical

9.2.4. Angular
version
Hyperbolic
ions
Special functions
Constants

Previous topic
9.1. `numbers` — Numeric abstract base classes

Next topic
9.3. `cmath` — Mathematical functions for complex numbers

This Page
Report a Bug
Show Source

9.2. `math` — Mathematical functions

`math.`**`ceil`**(*x*)

Return the ceiling of *x*, the smallest integer greater than or equal to *x*.

Th functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

9.2.1. retic and representation functions

float, delegates to `x.__ceil__()`, which should return an `Integral` value.

`math.`**`copysign`**(*x*, *y*)
Return a float with the magnitude (absolute value)
-1.0.

`math.`**`fabs`**(*x*)
Return the absolute value of *x*.

`math.`**`factorial`**(*x*)
Return *x* factorial. Raises `ValueError` if *x* is not inte

`math.`**`floor`**(*x*)
Return the floor of *x*, the largest integer less than o
value.

`math.`**`fmod`**(*x*, *y*)
Return `fmod(x, y)`, as defined by the platform C lib
C standard is that `fmod(x, y)` be exactly (mathema
same sign as *x* and magnitude less than `abs(y)`. Pyt
float arguments. For example, `fmod(-1e-100, 1e100)` is `-1e-100`, but the result of Python's `-1e-100 % 1e100` is `1e100-1e-100`, which cannot be

- This is a **specification**
  - Enough info to **call** function
  - But not how to **implement**
- Write them as **docstrings**

Anatomy of a Specification

One line description, followed by blank line

```
def greet(n):
    """Prints a greeting to the name n

    Greeting has format 'Hello <n>!' Followed by
    conversation starter.

    Parameter n: person to greet Precondition:
    n is a string""" print('Hello '+n+'!')
    print('How are you?')
```

Anatomy of a Specification

```
def greet(n):
    """Prints a greeting to the name n

    Greeting has format 'Hello <n>!
    conversation starter.

    Parameter n: person to greet
    Precondition: n is a string""" print('Hello
    '+n+'!') print('How are you?')
```

One line description, followed by blank line

More detail about the function. It may be many paragraphs.

Anatomy of a Specification

```
def greet(n):
    """Prints a greeting to the name n

    Greeting has format 'Hello <..
    conversation starter.

    Parameter n: person to greet
    Precondition: n is a string""" print('Hello
    '+n+'!') print('How are you?')
```

One line description, followed by blank line

More detail about the function. It may be many paragraphs.

Parameter description

Anatomy of a Specification

```
def greet(n):
    """Prints a greeting to the name n

    Greeting has format 'Hello <n
    Followed by conversation star

    Parameter n: person to greet
    Precondition: n is a string"""
    print('Hello '+n+'!') print('How
    you?')
```

One line description, followed by blank line

More detail about the function. It may be many paragraphs.

Parameter description

Precondition specifies assumptions we make about the arguments

Anatomy of a Specification

```
Def to_centigrade(x):
    """Returns: x converted to centigrade

    Value returned has type
    float.
    Parameter x: temp in
    fahrenheit Precondition: x
    is a float""" return 5*(x-
    32)/9.0
```

One line description, followed by blank line

More detail about the function. It may be many paragraphs.

Parameter description

Precondition specifies assumptions we make about the arguments

Anatomy of a Specification

```
def to_centigrade(x):
    """Returns: x converted to centigrade

    Value returned has type
    float.
    Parameter x: temp in
    fahrenheit Precondition: x
    is a float""" return 5*(x-
    32)/9.0
```

“Returns” indicates a fruitful functions

More detail about the function. It may be many paragraphs.

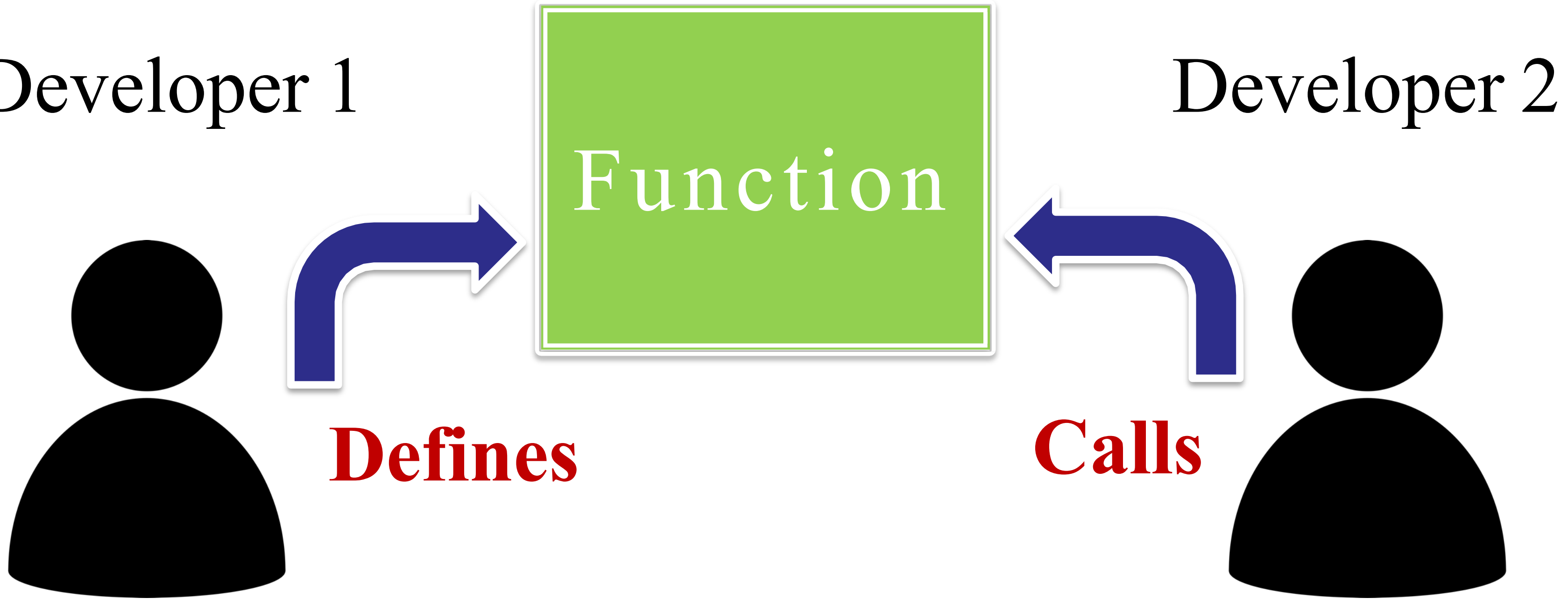
Parameter description

Precondition specifies assumptions we make about the arguments

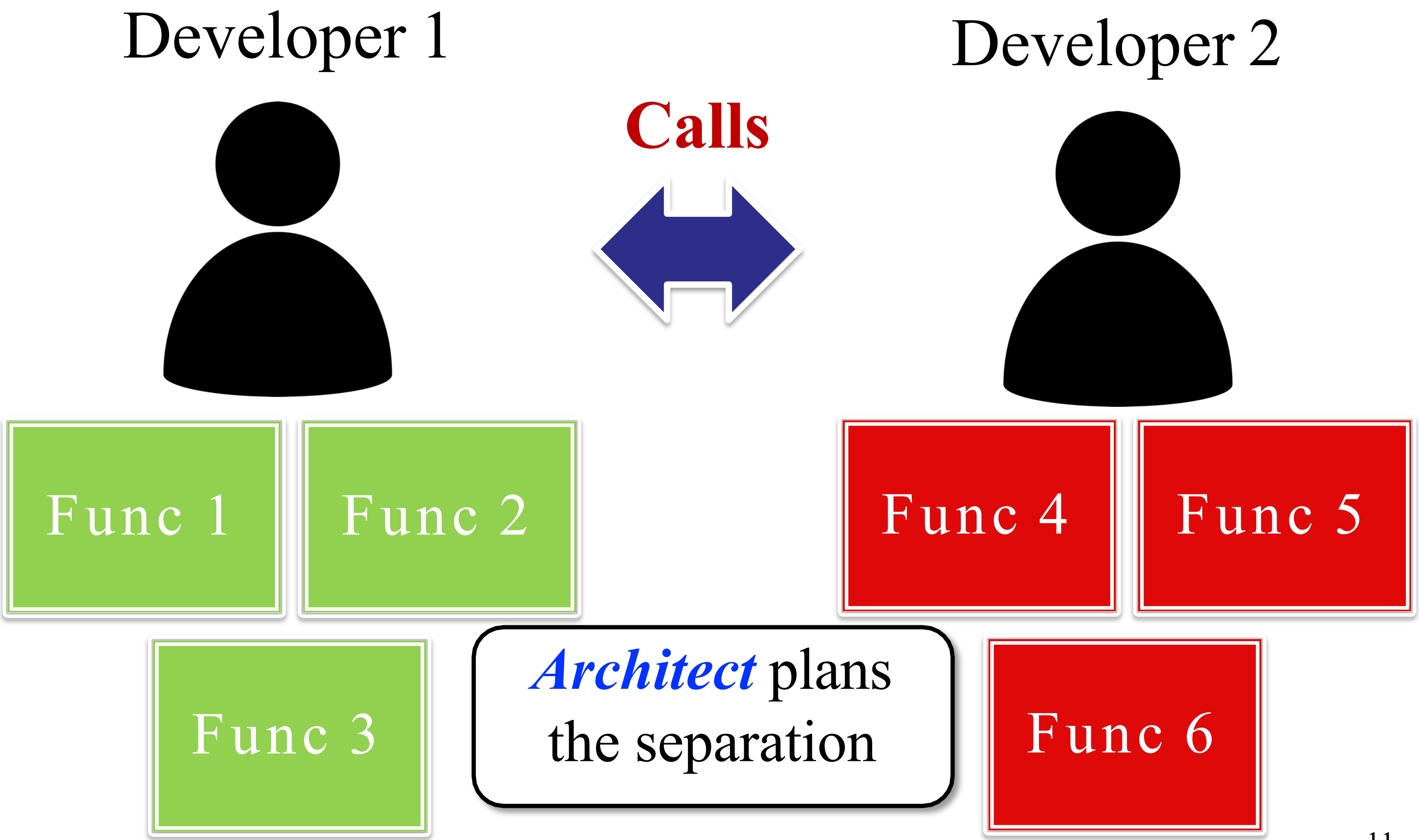
What Makes a Specification "Good"?

- Software development is a **business**
  - Not just about coding – **business processes**
  - Processes enable better code development
- Complex projects need **multi-person** teams
  - Lone programmers do simple contract work
  - Teams must have people working separately
- Processes are about how to **break-up** the work
  - What pieces to give each team member?
  - How can we fit these pieces back together?

Functions as a Way to Separate Work

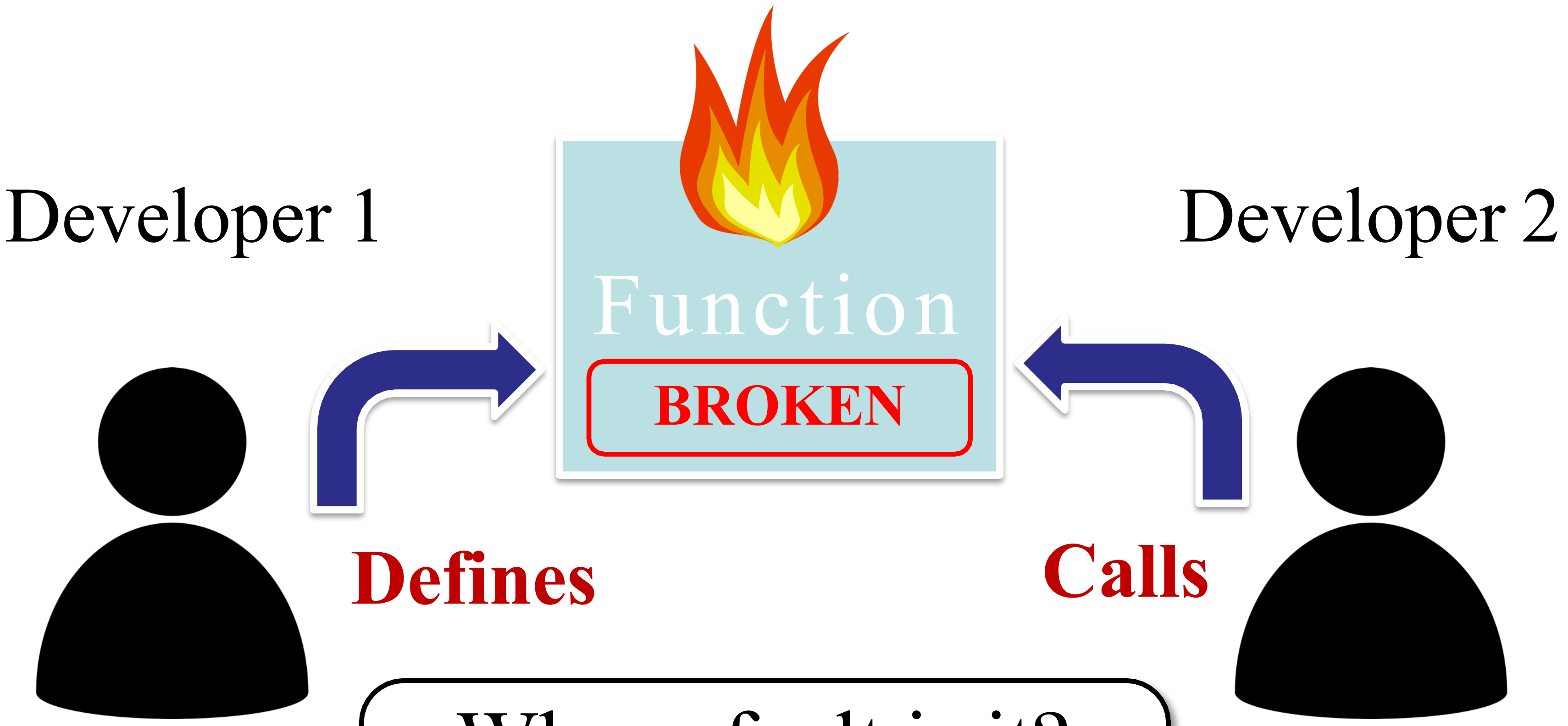


Working on Complicated Software



Architect plans the separation

What Happens When Code Breaks?



Whose fault is it?
Who must fix it?

Purpose of a Specification

- To clearly layout **responsibility**
  - What does the function promise to do?
  - What is the allowable use of the function?
- From this responsibility we determine
  - If definer implemented function properly
  - If caller uses the function in a way allowed
- A specification is a **business contract**
  - Requires a formal documentation style
  - Rules for modifying contract *beyond course scope*

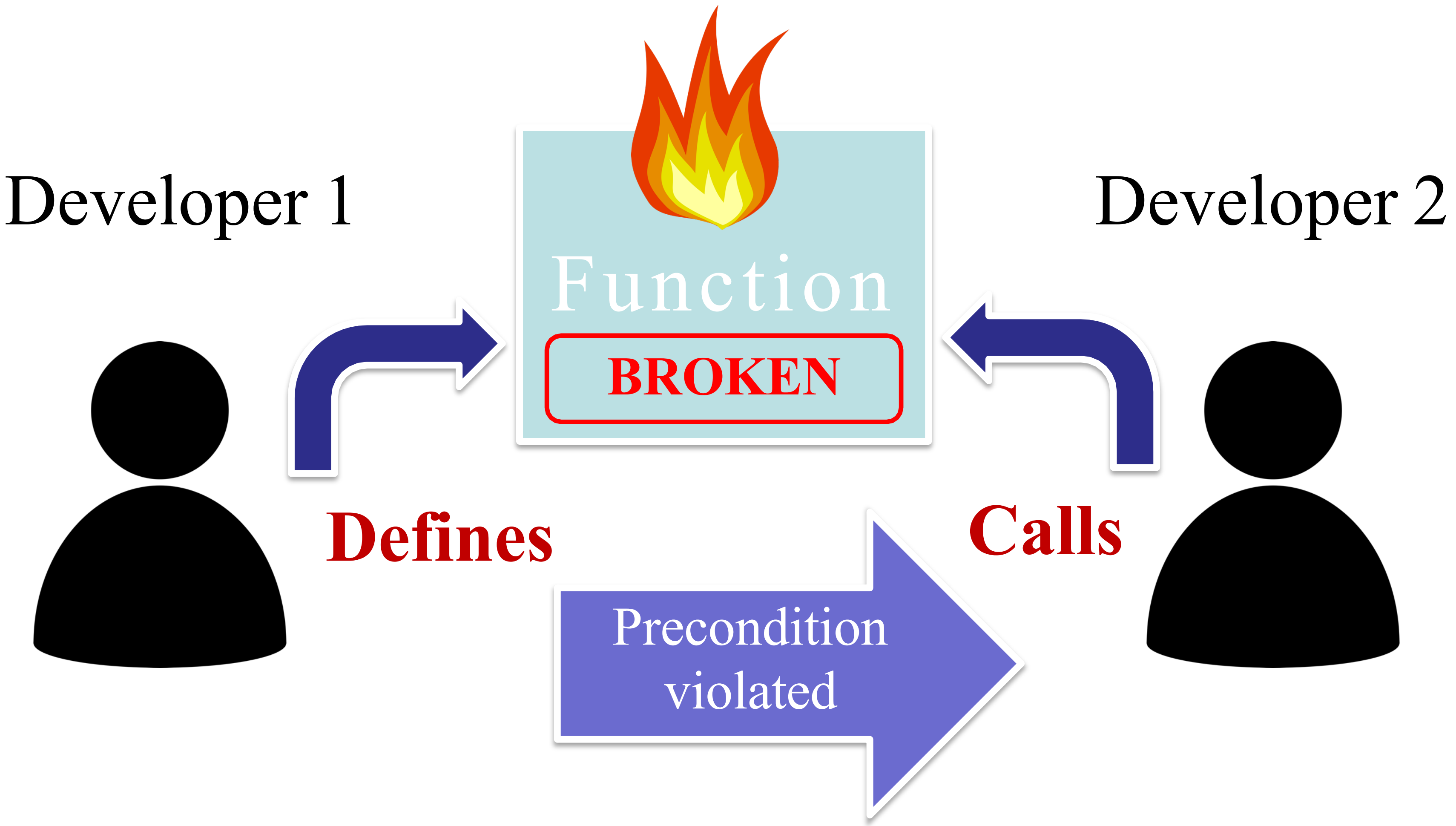
Preconditions are a Promise

- If precondition true
  - Function must work
- If precondition false
  - Function might work
  - Function might not
- Assigns responsibility
  - How to tell fault?

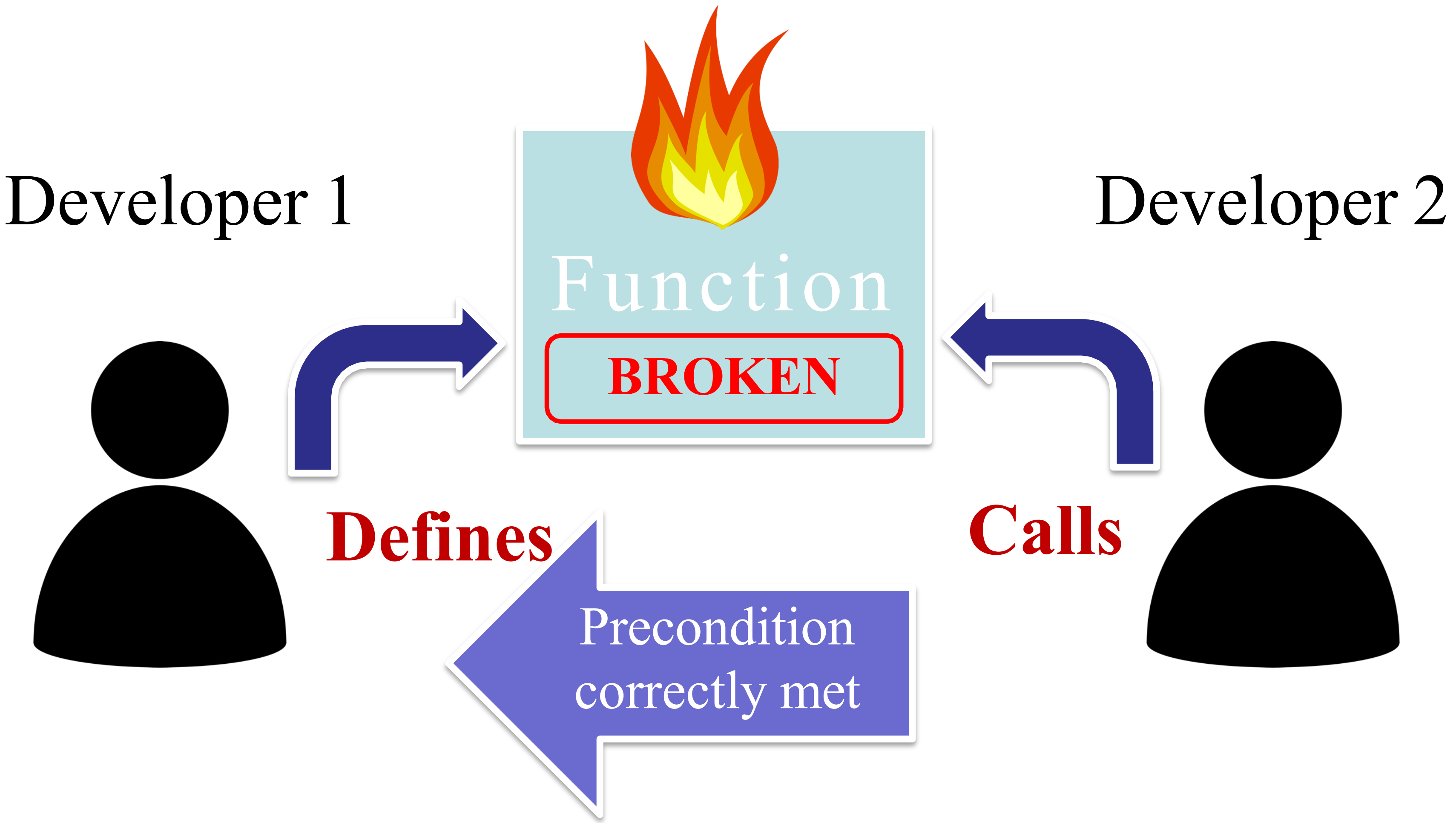
```
>>> to_centigrade(32.0)
0.0
>>> to_centigrade('32')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "temperature.py", line 19 ...
TypeError: unsupported operand type(s)
for -: 'str' and 'int'
```

Precondition violated

Assigning Responsibility



Assigning Responsibility



What if it Just Works?

- Violation != crash
  - Sometimes works anyway
  - *Undocumented* behavior
- But is **bad practice**
  - Definer may change the definition at any time
  - Can do anything so long as specification met
  - Caller code breaks
- Hits Microsoft devs a lot

```
>>> to_centigrade(32.0)
0.0
>>> to_centigrade(212)
100.0
```

Precondition violated

Precondition violations are unspecified!

Testing Software

- You are **responsible** for your function definition
  - You must ensure it meets the specification
  - May even need to prove it to your boss
- **Testing**: Analyzing & running a program
  - Part of, but not the same as, **debugging**
  - Finds **bugs** (errors), but does not remove them
- To test your function, you create a **test plan**
  - A test plan is made up of several **test cases**
  - Each is an **input** (argument), and its expected **output**

Test Plan: A Case Study

```
def number_vowels(w):
    """
    Returns: number of vowels in string w.

    Parameter w: The text to check for vowels
    Precondition: w string w/ at least one letter and only
    letters """
    …
```

Brainstorm
some test cases

Test Plan: A Case Study

```
def number_vowels(w):
    """
    Returns: number of vowels in
    string w.
    Parameter w: The text to check for vowels
    Precondition: w string w/ at least one letter and
    only letters """
    …
```

rhythm?
crwth?

Surprise!
Bad Specification

Test Plan: A Case Study

```
def number_vowels(w): """
  Returns: number of vowels in string w.

   Vowels are defined to be 'a','e','i','o', and 'u'. 'y' is a vowel if it is
   not at the start of the word.

   Repeated vowels are counted separately.Both upper case and
   lower case vowels are counted.

   Examples: ….

   Parameter w: The text to check for vowels
   Precondition: w string w/ at least one letter and only letters """
```

Test Plan: A Case Study

```
def number_vowels(w):
  """
  Returns: number of vowels

  Vowels are defined to be 'a' ,
  not at the start of the word

  Repeated vowels are counted separately.        Both upper case
   and lower case vowels are counted.

   Examples: ….
  Parameter w: The text to check for vowels
  Precondition: w string w/ at least one letter and only letters
  """
```

Some Test Cases

INPUT	OUTPUT
'hat'	1
'aeiou'	5
'grrr'	0

Representative Tests

- We cannot test all possible inputs
  - "Infinite" possibilities (strings arbitrary length)
  - Even if finite, way too many to test
- Limit to tests that are **representative**
  - Each test is a significantly different input
  - Every possible input is similar to one chosen
- This is an art, not a science
  - If easy, no one would ever have bugs
  - Learn with much practice (and why teach early)

Representative Tests

Simplest case first!

A little complex

"Weird" cases

Representative Tests for
`number_vowels(w)`

- Word with just one vowel
  - For each possible vowel!
- Word with multiple vowels
  - Of the same vowel
  - Of different vowels
- Word with only vowels
- Word with no vowels

How Many “Different” Tests Are Here?

number_vowels(w)

INPUT	OUTPUT
'hat'	1
'charm'	1
'bet'	1
'beet'	2
'beetle'	3

A: 2
B: 3
C: 4
D: 5
E: I do not know

How Many “Different” Tests Are Here?

number_vowels(w)

INPUT	OUTPUT
'hat'	1
'charm'	1
'bet'	1
'beet'	2
'beetle'	3

A: 2
B: 3 **CORRECT(ISH)**
C: 4
D: 5
E: I do not know

- If in doubt, just add more tests
- You are never penalized for too many tests